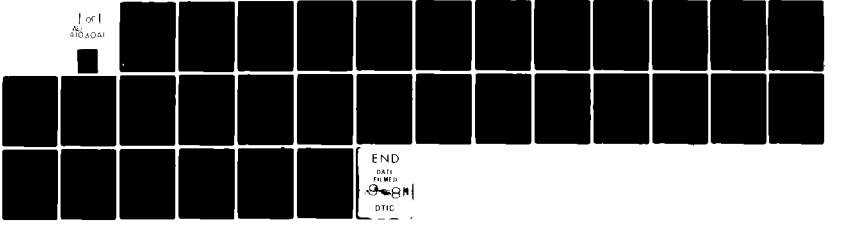
AD-A103 041 ILLINOIS UNIV AT CHICAGO CIRCLE F/G 5/1
DATABASE ALERTING TECHNIQUES FOR OFFICE ACTIVITIES MANAGEMENT,(U)
JUN 81 J CHANG, S CHANG N00014-80-C-0651
UNCLASSIFIED NL



Database Alerting Techniques for Office Activities Management

Jo-Mei Chang
Bell Telephone Laboratories
Murray Hill, N. J. 07974

and

Shi-Kuo Chang*
University of Illinois at Chicago Circle
Chicago, IL. 60680

First Draft May 10, 1980
Revised June 18, 1981

Abstract: In this paper, we approach the problem of office activities management from the database viewpoint. Database alerting techniques are developed to serve the purpose of office activities management. A conceptual framework for office information system design is presented. Simple database alerters and implementation techniques, existential alerters and time alerters are discussed. An example of journal editing is described in detail to clarify concepts. Finally, alerter system stability is discussed.

Keywords: office automation, office information systems, database alerting techniques, office activities management, knowledge-based database systems

*This author's research was supported by Office of Naval Research under ONR contract N00014-80-C-0651.

1. Introduction

Office automation can be defined as the replacement of manual office activities by identical or similar activities which automate means of doing office work. An office activity refers to any activity in an office, such as filling out a form, sending a message, entering information into a file, making a decision to route a form, etc. An office procedure refers to a structured set of office activities for the accomplishment of a specific office task, such as scheduling a meeting, processing a mortgage application form, reviewing a paper, etc. An office usually consists of a number of work stations or simply stations. The work stations are interconnected by a communication network to serve as a message exchange system.

The first step in office automation is usually the partial (or total) replacement (or enhancement) of the office desks by terminals, word-processors or small computer systems which are interconnected by an electronic message exchange system. In other words, initially office automation aims at the automation of devices and improvement of the communication network (message exchange system).

The second step in office automation is to take advantage of the electronic desks, and replace some manual office procedures by computerized procedures. At some work stations, manual intervention is still necessary for data entry and decision-making. Some work stations may be partially automated, with some activities managed by the electronic desk. Some other work stations may be entirely automated using computerized office procedures.

Since the first and ultimate goal of office automation is to automate

office activities, it is important to understand the problems related to office activities management. In this paper, we will approach the problem of office activities management from the database viewpoint. Database alerting techniques are developed to serve the purpose of office activities management. A conceptual framework for office information system design is presented in Section 2. Section 3 discusses simple database alerters and implementation techniques. Existential alerters and time alerters are discussed in Section 4. In Section 5, an example of journal editing is described in detail to clarify concepts. In Section 6, alerter system stability is discussed. Some concluding remarks are given in Section 7.

2. A Conceptual Framework for Office Information System Design

2.1. Event Monitoring Using Database Alerters

One of the major problems in office automation is the coordination and integration among various tasks. In the real world, actions are usually triggered due to a change of state of a certain event. Some of these actions are time-related routine operations, for example, routing a meeting notice among a group of people. Such routine operations are often periodically scheduled. Some of these actions are predetermined, for example, managing an editorial office which requires the coordination of a number of predetermined tasks. Such predetermined actions are usually both event-based and time-related. When we consider the design of an office information system, the monitoring of events, and the scheduling of predetermined and time-related routine activities are the main functions that an office information system can perform and thereby improve office efficiency and increase

productivity.

An office information system requires the support of a database management system for the storage and manipulation of office information. Moreover, the database management system should also be capable of responding to external events. Database systems are usually passive, in the sense that they only respond to externally generated information retrieval/manipulation requests, but cannot take other actions spontaneously. The recent development of database alerting techniques have changed the character of the database system from a passive one to an active one. Database alerters are first introduced by Buneman [BUNEMAN77, 79]. To clarify the concept, consider the following examples:

> Example 1: "Report the name and temperature of any station at which the temperature falls below ten degrees Centigrade."
>
> Example 2: "Report the number and owner of any account from which more than $500 is drawn."

In each of the above example, the user wishes to be informed when certain exception condition occurs. The exception condition and the prescribed message sending action form a <condition, action> pair. Such rules are called alerter rules.

Alerter rules can be used to monitor state transitions in a database. The current contents of a database determine its "state". If we aggregate the database states into two states, called the IN state (when an exception condition is met) and the OUT state (when an exception condition is not met), then an alerter is triggered and an alert message is generated or actions are invoked, whenever the database transits to an IN state. Such state changes are affected only by database updates. Therefore, to install database alerters, we need

only monitor database updates. The alerting subsystem in Figure 1 illustrates this concept.

Alerter rules therefore can be used to monitor database updates and trigger actions whenever certain conditions regarding database updates are satisfied. With alerter rules, database updates will automatically cause prespecified actions to take place. Thus the database system can take on an active role in events monitoring.

It should be noted that actions triggered by an alerter are viewed as side-effects of database updates. The failure of performing these actions does not necessarily cause the database updates to be rolled back. This constitutes the basic difference between alerters in an alerter system and triggers in an integrity system, such as the trigger mechanism proposed for system R [ESWAR76].

2.2. Activity Management in Office Information Systems

An office information system is a message-driven system. Work stations exchange messages, which cause certain office activities to be performed. In Figure 1, we present the component subsystems of an office information system.

Figure 1 depicts the relationship among the database system (DBMS), the alerting subsystem (AS), the office manager (OM), the activity agents (AA), and the intelligent coupler (IC). The user agent communicates with the user interface, called the intelligent coupler, by messages or forms. The intelligent coupler performs the translation of user queries [CHANG78b, 79]. It can also interact with the user to complete an intelligent form [ZISMAN77] (or interactive letter [ANDER76]).

The intelligent coupler sends user database retrieval and update messages to the database system. The database consists of a user database and an alerter database, both managed by the database system. The alerter database is used to store the alerter rules. The database system sends messages to the alerting subsystem, containing description of every completed update or failed update.

The alerting subsystem screens database updates to detect the occurrence of important events. Therefore, the alerting subsystem can be generally viewed as a screening program or a filter associated with the database system. When an event occurs, the alerting subsystem can either send an alert message to the intelligent coupler (which in turn informs the user agent), or send an activity request to the office manager to initiate office activities.

The office manager manages office activities. It receives activity request either indirectly from the alerting subsystem, or directly from the intelligent coupler. The office manager then schedules and performs office procedures by calling upon one or many activity agents. Therefore, the office manager is functionally analogous to the scheduler in an operating system.

An activity agent can be regarded as an office specialist, capable of performing some well-defined office activity [ZISMAN77]. For example, the activity agent can be a form generation program, a report generator, an editing program, etc. When an activity agent completes its task, it sends an activity completion message to the office manager, which may then schedule other activities.

The activity agent can send messages to the intelligent coupler, which

then presents the information to the user agent. The activity agent can also perform database retrieval/update operations, which may lead to triggering of alerters and scheduling of additional activities.

The alerting subsystem, the office manager, and the activity agents, together form the activity management system (AMS), which monitors events and initiates, schedules, and performs office activities [CHANG80]. The activity management system is therefore the most important part of the office information system.

2.3. Office Procedure Model

To describe the relationships among office activities, databases, and alerters, we adopt the following formalism: a rectangular box is used to denote a file Ri, a diamond-shaped box an alerter rule Aj, and a circle any activity or process Pk. If an activity or process requires user interaction, it is denoted by a double circle. A triangle is used to denote a message ui or a form Fj. An arrow leading to the base of a triangle indicates message reception, and an arrow emanating from the vertex of a triangle indicates message transmission. The possible relationships are summarized as follows:

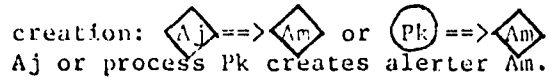
(1) File access: [Ri] -->> (Pk)
File Ri is accessed by process Pk.

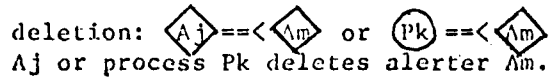
(2) File update: (Pk) -->> [Ri] or <Aj> -->> [Ri]
File Ri is updated by process Pk or alerter rule Aj.
We say that Pk or Aj affects file Ri.

(3) Alerter triggering: [Ri] --> <Aj>
Update of file Ri may trigger alerter Aj. The alert condition can be written beneath the directed arc. We say that Aj monitors file Ri for triggering.

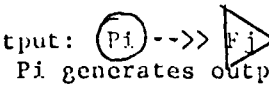
(4) Activity invocation: <Aj> --> (Pk)
Alerter Aj invokes process Pk.

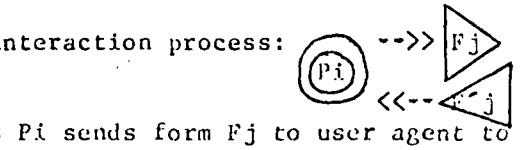
(5) Activity precedence: (Pk) --> (Pm)
Process Pk precedes (and invokes) process Pk.

(6) Alerter creation: 
Alerter Aj or process Pk creates alerter Am.

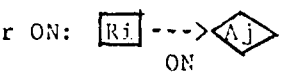
(7) Alerter deletion: 
Alerter Aj or process Pk deletes alerter Am.

(8) Message input: 
Input message u is stored in a message file.

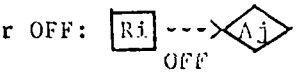
(9) Form output: 
Process Pi generates output form Fj.

(10) Form interaction process: 

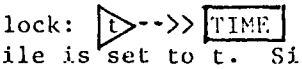
Process Pi sends form Fj to user agent to obtain additional information. When form Fj is completed by user, process Pi continues.

(11) Alerter ON: 

Update of file Ri may enable alerter Aj. The ON condition can be written beneath the directed arc. We say that Aj monitors file Ri for ON condition.

(12) Alerter OFF: 

Update of file Ri may destroy alerter Aj. The OFF condition can be written beneath the directed arc. We say that Aj monitors file Ri for OFF condition.

(13) Time clock: 
TIME file is set to t. Since TIME file is often a conceptual device (see Section 4), the update of TIME file can be omitted in an OPM specification.

With these formal notations, we can graphically depict office procedures and analyze how they can be executed by an office information system. Such a formal model is called the office procedure model (OPM). This knowledge model can also be stored in the database. It is accessed by the alerting subsystem to check for alerter database consistency. It is also accessed by the office manager to schedule and control concurrent activities invoked by the office manager, and to check for alerter system stability (see Section 6).

3. Simple Alerters and Implementation Issues

Simple alerters monitor database updates of simple database objects, usually records in a file (or tuples in a relational file, if we use the relational database terminology). To specify a simple alerter, we need to specify the following:

(1) Name of alerter: This is a unique symbolic name to identify an alerter.

(2) Type of update operation to be monitored: For updates of records in a file, there are three types of updates: insertion of a new record, deletion of an old record, and modification of an old record. The three update types are denoted by "i", "d", and "m", respectively.

(3) Name of database object to be monitored: For monitoring of record updates in a file, this will consist of two parts:

(3.1) file name (or relation name), and

(3.2) field names (or attribute set).

(4) Alert condition: The alert condition is a logical expression involving atomic clauses. Each atomic clause consists of an attribute name and a literal, or two attribute names, related by a comparison operator such as "=", "!=", "<", ">", etc. In an alert condition for type "m" update (record modification), the attributes are prefixed by "old" or "new", indicating whether an attribute refers to the "old" or the "new" record. Similarly, in an alert condition for type "i" update (record insertion), the attributes are prefixed by the "new" keyword. In an alert condition for type "d" update (record deletion), the attributes are prefixed by the "old" keyword. In these two cases, however, the prefix can be omitted since there is no possibility for

confusion.

(5) Action: The action taken by a simple alerter when it is triggered, is to send messages to various users or to invoke another process, or to perform database update operations.

(6) Name of creater of the alerter rule.

Alerter rules are stored in the alerter database (ADB), which is also managed by the database system. Referring again to the examples mentioned in Section 2.1, the messages to create the appropriate alerters are:

```
(1) ADDALERT  a-name="frostwarning", u-type="m",
              rel-name="weather",
              attribute-name="temp", condition="new.temp<10",
              action="ALERT user-a user-b",
              creator="user-c"
```

(The alerter name is "frost-warning", update type is "m", relation name is "weather", attribute is "temp", condition is "new.temp<10", alert message should be sent to "user-a" and "user-b", and alerter is created by "user-c".)

```
(2) ADDALERT  a-name="withdrawn-warning", u-type="m",
              rel-name="account",
              attribute="balance",
              condition="old.balance - new.balance > 500",
              action="ALERT bank-manager",
              creator="teller-a"
```

(The alerter name is "withdrawn-warning", update type is "m", relation name is "account", attribute is "balance", condition is "old.balance-new.balance>500", alert message should be sent to "bank-manager", and alerter is created by "teller-a".)

An alerter can be removed by a deletion message:

DLTALERT "frostwarning"

Database retrieval/update requests from the user agent are sent to the database system. The ADDALERT and DLTALERT messages, on the other hand, are sent to the alerting subsystem, which then uses the database system to perform the actual updating of the alerter database.

When the alerting subsystem receives an ADDALERT message, it adds the appropriate alerter rule to the ADB, after checking that the rule is acceptable (e.g. the database object does exist, and the rule is consistent with other rules). A message

ADDEDALT alerter-name

is sent to the agent creating the alerter rule, where "alerter-name" is the symbolic name of the alerter rule. If the alerting subsystem finds the alerter rule unacceptable, an appropriate error message is returned.

Similarly, when the alerting subsystem receives a DLTALERT message, it deletes the specified alerter rule from the ADB and sends the following response to the agent deleting the alerter rule,

DLTEDALT alerter-name

When a database retrieval request is sent to the database system, it simply retrieves the appropriate information, and the response from the database system is forwarded to the user. No message is sent to the alerting subsystem. This is because we do not monitor retrieval operations. Retrieval could be monitored, if we intend to analyze user profile for security or protection reasons.

When a database update request is sent to the database system, it first performs the requested update operation. Only after the update has been performed, that the database system sends the following message,

UPDATED <update-type> <obj-name> <old-record> <new-record>

to the alerting subsystem. The alerting subsystem then checks whether any alert condition is satisfied. The simplest approach is to scan through the alerter rules in the alerter database. The alerting subsystem can use the database system to retrieve alerter rules from the alerter database. To improve efficiency, the alerter rules could be indexed by: (a) update type, (b) relation name, and (c) attribute name(s). With such an index structure, the lowest-level entries are pointers to the alerter rules. Only alerter rules pertinent to an update need be checked. Therefore, in practice, the required computation for checking ADB represents a small overhead on each update.

When an alerter is triggered, the alerting subsystem may send alert messages to the user agent, if the specified action is the ALERT command. The alert message contains the name of alerter, type of update, database object monitors, and value of database object before and after the update. The user agent is responsible for processing the ALERT message. On the other hand, the alerting subsystem may send activity request to the office manager for activity scheduling. The activity request consists of action names and other parameters, as specified in the alerter rules.

4. Existential Alerters and Time Alerters

Existential alerters are alerters with well-defined duration. To define duration of an alerter, we extend the concept of simple alerters as follows. Each alerter is associated with three conditions: an alert condition, an ON condition, and an OFF condition. When the ON condition is met, the alerter is enabled. When the OFF condition is met, the alerter is destroyed. The alert condition, as defined previously, determines when the alerter rule is triggered. Each alerter thus monitors three database objects, one for each of the three conditions. These three monitored objects can be identical or different. The set of alerters that are currently ON is called the ON-set.

A customized alerter is an alerter of the form Ak(c1, c2, ..., cm) where the ci's are parameters that may appear in the alert condition. A customized existential alerter is an existential alerter of the form Ak(c1, c2,..., cm), where the ci's are parameters that may appear in the alert condition or the duration (i.e, the ON condition and the OFF condition). All customized alerters of an alerter Ak have the same format as Ak, except the ci's may have different values in each customized rule.

A time alerter is a special type of alerter for monitoring time-related events. In order to specify time alerters, we can assume there is a special relational file, called TIME, which has only one attribute -- time. The system may update the TIME relational file periodically, the update frequency being dependent on applications. The TIME relational file may also be the system clock itself, and the update frequency is the same as the clock rate. With this conceptual

time relational file, the system can treat time as an ordinary attribute, and the alerter rule can mention the time attribute in its ON condition, trigger condition, as well as OFF condition.

As an example, suppose we want to monitor incoming telephone calls. The alerter is in effect between 8 a.m. and 11 a.m., and the trigger condition is "caller = 'Smith'". The existential alerter is as follows:

(1) ON condition: time = 8 a.m.
(2) Alert condition: caller = 'Smith'
(3) OFF condition: time = 11 a.m.

For this application, the TIME file might be updated once every five minutes.

The above alerter rule can be modified to be a customized alerter rule as follows. The customized alerter A(t1, t2, caller-name) has the following conditions:

(1) ON condition: time = t1
(2) Alert condition: caller = caller-name
(3) OFF condition: time = t2

Other system parameters can be monitored similarly, by creating special-purpose system relational files containing such parameters. The system overhead is proportional to the frequency of updates for such system files, because every update of a system file will result in the evaluation of alerter rules monitoring this file. Therefore, we must exercise care in determining how often to update the system parameters, such as time, toggle switch, etc.

5. Journal Editing Example

As an example of office automation, we will describe the journal editing activities in an editorial office. Basically, there are three activities to be considered: (a) occurrence of real-world events, (b) database update activities, and (c) generation of forms.

The occurrence of real-world events causes input messages to be sent to the office information system. As illustrated in Figure 2, each input message is considered a record insertion into a MESSAGE relational file in the user database UDB, which triggers alerters A1 or A6 to invoke user-defined processes. In actual implementation, the message file may be nonexistent, or it may serve as a log file to record all incoming messages. The message file may contain the following attributes: message-type, message-id, message-text.

One type of input message is the submission of a paper from an author. This message, with message-type 's', represents an event which arises from outside the system. It triggers A1 to invoke a (manual or automatic) data entry process P1 to enter the relevant information into the user database. In this case, a new record is inserted into the PAPERS relational file. The PAPERS file has the following attributes: paper#, title, author, author-address, submission-date, paper-status.

The insertion of a new record in PAPERS relational file triggers alerter A2 to invoke two concurrent processes: (1) a form generation process P2 to send an acknowledgement letter (form F2) to the author; and (2) a reviewer selection process P3 to prompt the editor to select three reviewers. The form generation process P2 is automatic. The process P3 requires manual interaction. The interaction is accomplished using a form F3.

After the editor has selected three reviewers using form F3, process P3 causes the insertion of a new record into the REVIEW relational file. The REVIEW file has the following attributes: paper#, reviewer1, date1, st1, reviewer2, date2, st2, reviewer3, date3, st3. The status of a reviewer is initially 0. It is set to 1 when the reviewer sends back the review, and -1 when he declines to review the paper. The insertion of a new record in the REVIEW relational file triggers alerter A3 to invoke a form generation process P4, to send letters (form F4) and copies of the submitted paper to the reviewers. The reviewer's name and address can be found in another relational file REVIEWER, which contains the following attributes: reviewer#, name, address, review-area.

The alerter A3 also generates an existential time-alerter A4(X,Y,t1), for reviewer X, paper Y, at time t1. When a reviewer has not responded after a given time interval (say, three months), A4 is triggered by the alerting subsystem. A4 invokes a form generation process P5, to send a letter (form F5) to that reviewer asking for response. The alerter A4 generates another existential time alerter A5(X,Y,t2) and then self-destructs. It should be noted that A4 is an existential alerter which is automatically destroyed when the reviewer send back his review.

If the reviewer still does not respond after a given time interval, A5 is triggered, which again invokes process P3. The process P3 prompts the editor to select another review, and again updates the REVIEW relational file. After firing, the alerter A5 also self-destructs. A5 is also an existential alerter which is automatically destroyed when the reviewer sends back his review.

Another type of input message occurs when a reviewer sends back his review. Again, this message, with message-type 'r', is considered as an insertion into MESSAGE relational file, which triggers alerter A6 to invoke a (manual or automatic) updating process P6, to update the REVIEW relational file. The update of REVIEW may cause the destruction of existential alerters A4 and A5. If all three reviews of same paper have come back, (st1=st2=st3=1), this will trigger alerter A7, which invokes an evaluation process P7. P7 will require manual interaction with the editor to determine status of paper. The interaction again is accomplished using a form F7. A form F8 is generated, to inform the author that his paper is (a) accepted, (b) required to be revised, or (c) rejected. If paper is accepted or rejected, that record in PAPERS relational file may be moved to a backup file, and the corresponding record in REVIEW relational file may also be moved to a backup file. If paper is to be revised, it stays in PAPERS relational file, and the corresponding record in REVIEW relational file is updated.

If a reviewer sends back a letter, saying he does not want to review the paper, then this reviewer's status is changed to "-1", and the update of the REVIEW relational file triggers alerter A9, which again invokes process P3 to prompt the editor to select another reviewer, and the whole procedure repeats.

From the above description, we can see that messages can be created by the user to represent outside events, or by the system because of updating of relational files, user time interrupts, etc. Each message may trigger one or more alerters, which usually invoke processes to perform some of the following: (a) request additional information from

the user, (b) update the database, and (c) generate forms.

Figure 2 depicts a set of alerters to perform the journal editing task. There are two relational files: PAPERS and REVIEW. The MESSAGE file could be a nonexistent file, or a log file. Alerters A2 and A8 monitor the PAPERS relationla file, and A3, A7, and A9 monitor the REVIEW relational file. A4 and A5 are customized time alerters, and A5 is generated only when A4 has been triggered. A4 and A5 are both customized for a particular reviewer X of a particular paper Y, and they either self-destruct after firing, or are destroyed when OFF conditions are met, i.e. when the reviewer sends back his review. Figure 2 also depicts the relationship among various alerters. The notation introduced in the previous section is used, but duplicated relational files are drawn for the sake of clarity.

The above journal editing example illustrates the combination of manual and interactive activities (P1, P3, P6 and P7) with automated activities (P2, P4, P5 and P8). It also illustrates the usage of forms for office communications. Forms F2, F4, F5, and F8 are output forms. F3 and F7 are interactive forms, or so-called intelligent forms, which requires manual interaction. P1 and P6 are also interactive processes, because if the input messages are paper messages (such is the case in a conventional editorial office), then these input messages must be encoded and entered into the system. However, if we have an electronic mail system, then the paper submission message is a form F1, and the review update message is either the returned form F4 or F5, and in all these cases P1 and P6 are automatic processes. Notice also in Figure 2, A1, A4, A5, A6, A7 and A9 are conditional alerters. The other alerters do not have alert conditions.

Figure 3 illustrates the update of the alerter database (Figure 3(a)), the update of the PAPERS relational file (Figure 3(b)), the generated form F2 which is sent to the author (Figure 3(c)), and the interactive form F3 which is sent to the editor to be completed and returned to editorial office system (Figure 3(d)).

6. Alerter System Stability

We distinguish an alerting subsystem, which is the physical implementation of database alerting technique, from an alerter system, which is the abstract system of alerter rules driven by input updates. In this section, we discuss the problem of alerter system stability.

When alerters trigger each other in endless successions, infinite message loops occur. Such infinite message loops are obviously undesirable, because the alerter system is unstable. Therefore, infinite loop detection technique must be devised to prevent such infinite message loops from happening.

To illustrate the concept of infinite loop detection, let us consider the following example. Suppose the user data base has two relational files, R1(D11, D12) and R2(D21,D22). In the alerter data base, there are two alerter rules:

Alerter Rule A1: If a record (1, d12) in file R1 is modified to (1,1), then modify record (1, d22) in R2 to (1,2).

Alerter Rule A2: If a record (1, d22) in file R2 is modified to (1,2), then modify record (1, d12) in R1 to (1,1).

With these two alerter rules, when a record (1, d12) in R1 is modified to (1,1), the two alerter rules will be triggered successively, causing an infinite message loop.

Using the notation introduced in Section 2.3, in the example described above, we have R1 -> A1 ->> R2 -> A2 ->> R1, thus forming a message loop. Therefore, we say that alerter rules A(1), A(2), ..., A(n) form a message loop, if there are relational files R(1), R(2), ..., R(n),

such that R(1) -> A(1) ->> R(2) -> A(2) ->> ... ->> R(n) -> A(n) ->> R(1).

The existence of a message loop is a necessary condition for the occurrence of infinite message loops. However, whether an infinite message loop will occur is data dependent. As in the above example, if we insert record (0, 0) into relational file R1, no infinite message loop will occur.

Therefore, to prevent infinite message loops from occuring, it is necessary to (1) detect the presence of message loops, (2) keep a history of update messages and alerter rule firings for every relational file and alerter rule in the message loop, and (3) break up an infinite message loop when the update message histories indicate such loops are present.

The office procedure model (OPM) described in Section 2.3 is a graphic representation of the relationships among office activities, databases, and alerters. For each specific database update, we call the description of its relationship with the alerter rule triggered and activities performed a run-time OPM instance. The history of the alerting system actually consists of a collection of such run-time OPM instances. In the above infinite message loop detection scheme, Part (1) requires a static analysis of the OPM model to detect the message loop. Part (2) and Part (3) requires dynamic monitoring of OPM run-time instances within the message loop.

The detection of message loops can be achieved as follows. Each relational file can be represented as a node in a directed graph derivable from the OPM model. If there are Ri, Ai and Rj such that Ri->Ai->>Rj,

an arc will be drawn from Ri to Rj, with the arc labelled as Ai. The existence of a loop in the graph indicates the existence of a message loop among the alerter rules. Whenever a new alerter rule is added to the alerter database or an old alerter rule is updated, the system modify the OPM model, constructs the directed graph, and checks whether a message loop exists.

We now demonstrate that under certain conditions, infinite message loops can be detected. Let u, v denote records (or tuples) in a relational file R. An update operation is denoted by (u, v), where u is the record to be deleted, and v is the record to be inserted. If u is not in R, or v is already in R, the update operation (u, v) is undefined. An insertion operation is denoted by (0, v), and a deletion operation is denoted by (u, 0), where 0 denotes the empty set.

Suppose Ai monitors Ri. The trigger region Bi of Ai is defined to be,

Bi = { (u,v): u, v in Ri and (u,v) triggers Ai}

where Ri is the underlying domain of Ri.

Suppose Ai monitors Ri and Ai affects Rj. The notation, (ui, vi) -> Ai ->> (uj, vj) indicates that an update operation (ui, vi) on Ri will trigger Ai (i.e. (ui, vi) is in Bi), and Ai causes an update operation (uj, vj) on Rj.

If we denote (ui, vi) by wi and (uj, vj) by wj, we can simply write wi -> Ai ->> wj. In other words, we have a mapping f such that f(wi) = wj. Each component of the mapping is denoted by fk, and fk(wi) = wjk, where wjk is the k-th component of wj.

The alerter Ai is information reducing, if for every pair of (wi, wj)

such that wi -> Ai ->> wj, each component fk of the mapping f is either a generalized identity function (i.e. wjk = fk(wi) = wix for some x), or a finite classifier (i.e. the range of fk is finite).

If wi -> Ai ->> wj and Ai is information reducing, then wj has a finite range.

If there is an infinite message loop, then for some Ri -> Ai ->> Rj in a message loop, there are infinite sequences of update operations

$$w_i^1, w_i^2, w_i^3, \ldots\ldots w_i^n; \text{ and } w_j^1, w_j^2, w_j^3, \ldots\ldots w_j^n, \text{ such that}$$

$$w_i^k \text{ -> } A_i \text{ ->> } w_j^k$$

for all k. Since wi has finite range, we must have $w_i^M = w_i^N$ for some numbers *M* and *N*.

Therefore, we may wish to restrict the alerters to information reducing alerters. Under such constraints, we can dynamically determine whether an infinite message occurs by observing whether some wi recurs, for some Ri within a message loop.

We now describe a method to record the run-time OPM instance for message loop detection. Whenever the alerter database is updated, the activity management system checks whether a message loop has been formed. If a message loop is detected, additional alerter rules can be added automatically by the activity management system, one rule Ai for each relational file Ri in the message loop. Subsequently, whenever Ri is updated, Ai is always triggered to record the update message into a log file. The log file is a protected special relational

file in the UDB. The log file is indexed by (a) file name, and (b) user process-id. In other words, a history of update messages is kept for each (file name, user process-id) pair. The user process-id is the original user process-id which initiates the first update to a relational file in a message loop. In the above, we have shown that if the alerter rules are information reducing, then by observing whether there is a recurrence of update message in an update history, the occurrence of infinite message loops can be dynamically detected.

In practice, instead of keeping update message histories in the log file, we can keep a conunter for each (file name, user process-id) pair. Each file update will increment the counter by one. When a counter reaches a perdetermined threshold, the loop is broken by (a) blocking the update message having the same file name and original process-id, and (b) sending a report to the user.

Instead of keeping a log file or even a simplified log file as suggested above, we can include a frequency stamp in each message to store a count of how many times this message has looped through the activity management system. The activity management system is responsible for assigning proper values to the frequency stamp. The frequency stamp is usually inherited from the input message. When the input message does not have its frequency stamp specified, the activity management system will then assign an appropriate value to the frequency stamp, as to be described below. When the alert condition of an alerter rule is satisfied, the frequency stamp of the input message is incremented by one. If this value exceeds a preset threshold, the activity management system will break this loop by not triggering the alerter, and the user is notified. Otherwise the activity management

system will assign this frequency stamp to all output messages generated by this alerter rule.

If the messages are fedback directly to the activity management system, then the frequency stamps will be transmitted and incremented as described above. When the message is to activate an external process, we cannot expect the external process to transmit the frequency stamp. Therefore, whenever an alerter invoked an external process, the frequency stamp associated with this process is stored in a frequency stamp table, which contains the name of the process, and its frequency stamp. Whenever the activity management system receives an update message with an unspecified frequency stamp, it will consult the frequency stamp table to determine whether this message is caused by some alerter rule previously. If this is the case, then the process name associated with the current message will be a descendent process of some process recorded in the frequency stamp table. The current process should then inherit the frequency stamp of the ancestor process, and the unspecified frequency stamp is thus specified. Otherwise, the frequency stamp field of this message is set to the initial value 0.

Proper garbage collection work need to be performed to maintain the frequency stamp table. A process can be removed from the frequency stamp table when it ceased to be active. In practice, we can simply implement the frequency stamp table as a ring structure with some reasonable number of entries. For example, the number of entries may be equal to twice the maximum number of active processes allowed by the operating system. When a new entry is inserted into the frequency stamp table, this will eliminate the oldest entry from the table.

7. Discussions

In this paper, we propose to use database alerting techniques for office activities management. The role of an activity management system in an office information system is clarified, and techniques for mainitaining alerter system stability are discussed. A unified formalism to describe an office procedure model has been presented in Section 2.3. A diagram such as Figure 2 then represents the office activities at one work station. Similar diagrams can be constructed for other work stations, and together they can be used to represent a distributed office information system (DOIS). Such a model can be called a distributed office procedures model (DOPM).

An example distributed office procedure model for a three-node (or three-station) distributed office information system is illustrated in Figure 4. It can be seen that a distributed database management system (DDBMS) is needed to support a distributed office information system. Problems of infinite message loops, deadlocks, concurrency control, data consistency, and process conflicts must be analyzed carefully. Because of the complexity of distributed office information systems and the evolutionary nature of such systems, it is expected that more and more emphasis will be placed on the incorporation of knowledge (such as alerter rules, database skeletons [CHANG78a], and office procedure models) into such systems, so that it can function properly and provide adequate support for both routine activities management and decision-making.

References:

[ANDER76] R. A. Anderson and J. J. Gillogly, "Rand Intelligent Terminal Agent (RITA): Design Philosophy", Rand Report R-1809-ARPA, Rand Corporation, Santa Monica, CA, 1976.

[BUNEM77] O. P. Buneman and H. L. Morgan, "Implementing Alerting Techniques in Database Systems", Proc. of IEEE COMPSAC Conference, November 8-11, 1977, 463-469.

[BUNEM79] O. P. Buneman and E. K. Clemons, "Efficiently Monitoring Relational Databases", ACM Trans. on Database Systems, Vol. 4, No. 3, September 1979, 368-382.

[CHANG78a] S. K. Chang and W. H. Cheng, "Database Skeleton and its Application to Logical Database Synthesis", IEEE Transactions on Software Engineering, Vol. SE-4, No.1, January 1978, 18-30.

[CHANG78b] S. K. Chang and J. S. Ke, "Database Skeleton and its Application to Fuzzy Query Translation", IEEE Transactions on Software Engineering, Vol. SE-4, No. 1, January 1978, 31-44.

[CHANG79] S. K. Chang and J. S. Ke, "Translation of Fuzzy Queries for Relational Database System", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 3, July 1979, 281-294.

[CHANG80] J. M. Chang and S. K. Chang, "Database Alerting Techniques for an Activity Management System", Proceedings of 1980 International Computer Symposium, Taipei, Taiwan, Republic of China, December 1980.

[CONWA74] R. Conway, W. Maxwell and H, Morgan, "A Technique for File Surveillance", Proceedings of IFIP Congress 74, North-Holland Publishing Company.

[ELLIS79] C. A. Ellis, "Information Control Nets: A Mathematical Model of Office Information Flow", 1979 Conference on Simulation, Measurement and Modeling of Computer Systems, 225-239.

[ESWAR76] K. P. Eswaran, "Specifications, Implementations and Interactions of a Trigger Subsystem in an Integrated Database System", Technical Report RJ1820, IBM Research Laboratory, San Jose, California, August 1976.

[HAMME76] M. Hammer, "Error Detection in Data Base Systems", Technical Report, M.I.T. Laboratory for Computer Science, 1976.

[MCDON75] C. McDonald, B. Bhargava, D. Jeris, "A Clinical Information System for Ambulatory Care", Proc. of National Computer Conference, May 1975, Anaheim, California.

[TSICH80a] D. Tsichritzis, "Form Flow Models", Technical Report, University of Toronto, 1980.

[TSICH80b] D. Tsichritzis, "A Form Manipulation System", Technical Report, University of Toronto, 1980.

[ZISMAN77] M. D. Zisman, Representation, Specification and Automation of Office Procedures, Ph.D. Dissertation, University of Pennsylvania, 1977.

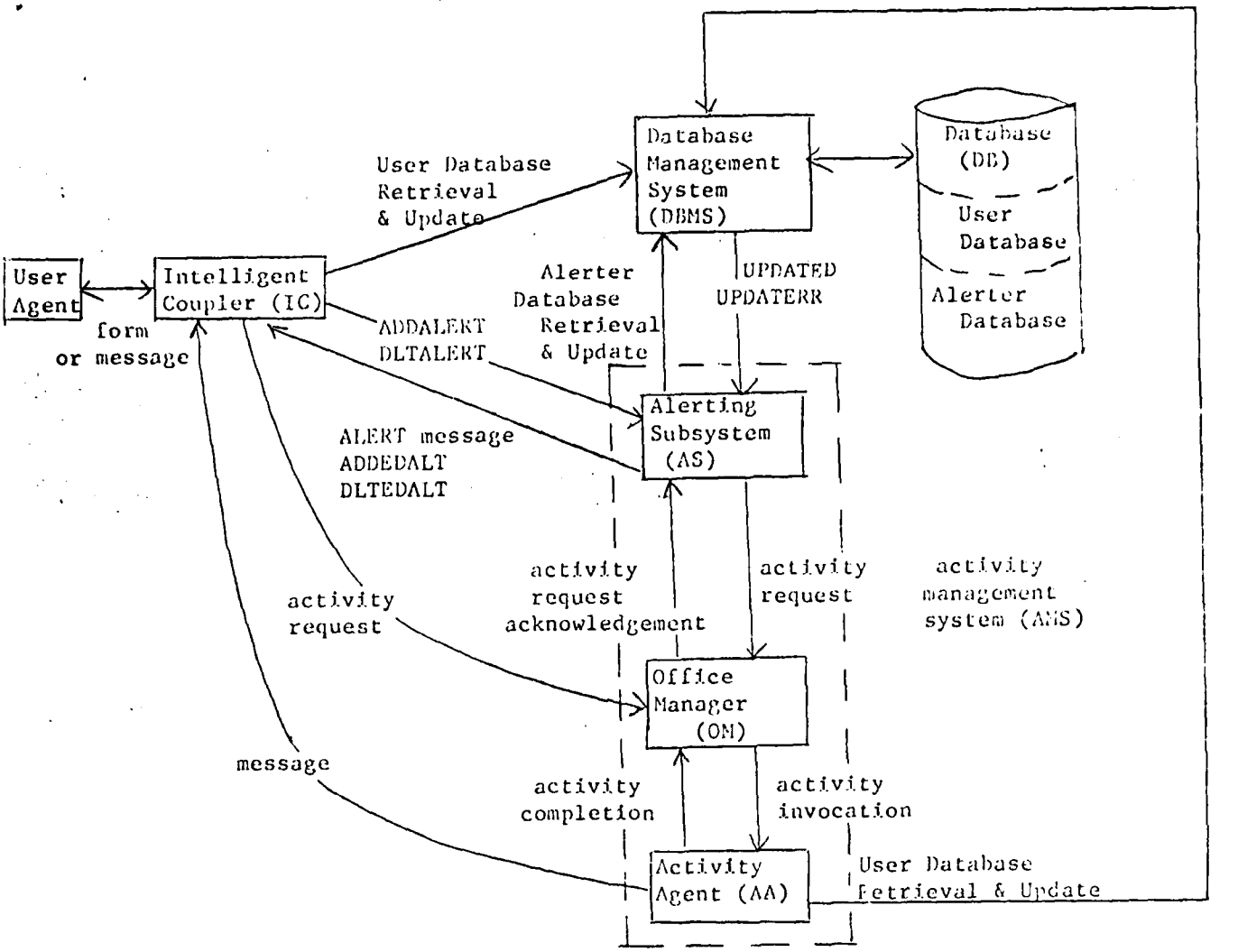


Figure 1 Component Subsystems of an Office Information System

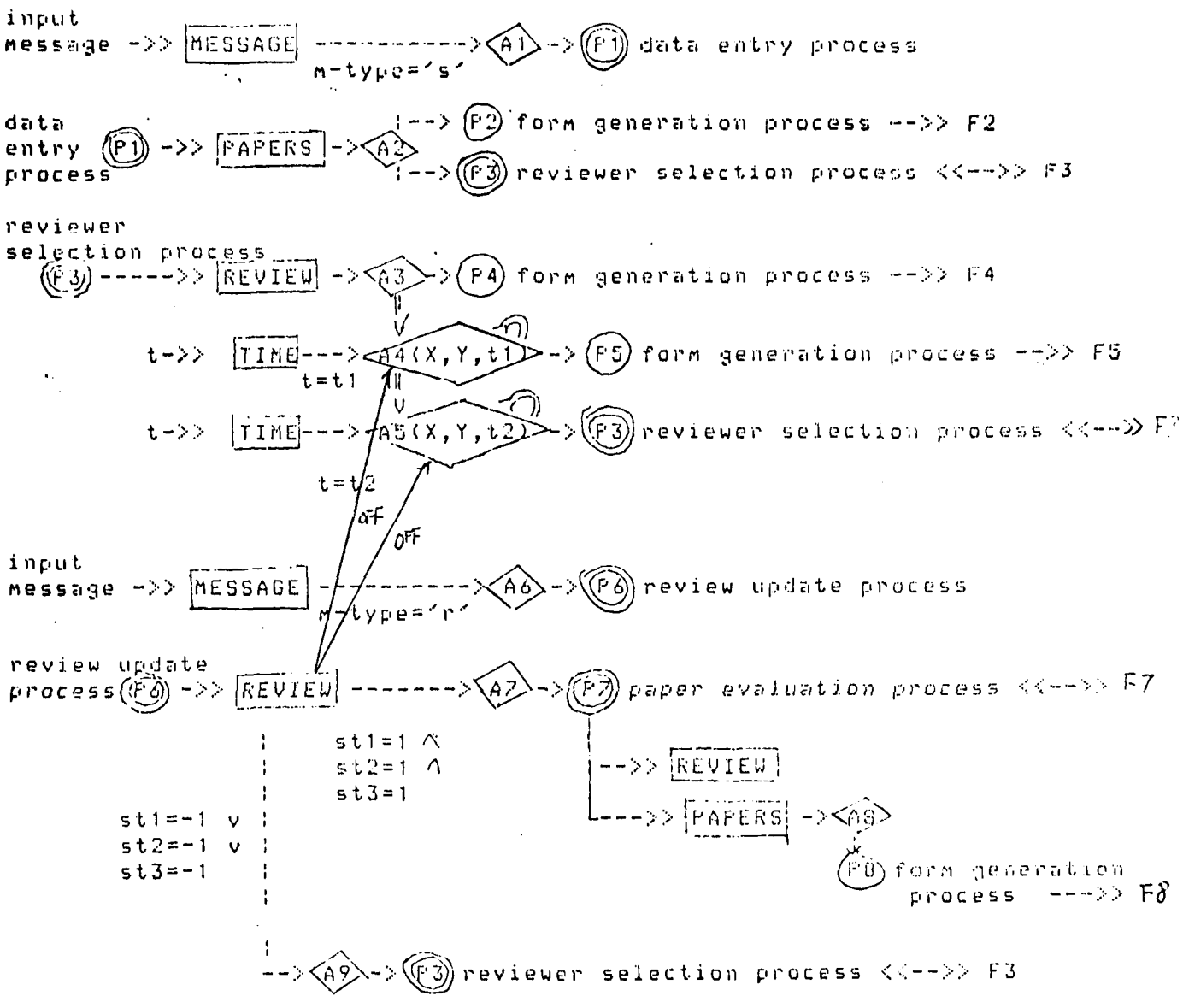


Figure 2 Office procedures model for the editorial office

```
ADDALERT  a-name="A2",u-type="i",rel-name="PAPERS",condition="",
          action="sendform F2 %author %author-address %title;
                  select-reviewer F3 %author %title",
          creator="Editor"

ADDALERT  a-name="A3",u-type="im",rel-name="REVIEW",condition="",
          action="sendform F4 %new.reviewer1 %new.paper#;
                  sendform F4 %new.reviewer2 %new.paper#;
                  sendform F4 %new.reviewer3 %new.paper#;
                  create-alerter A4 %new.reviewer1 %new.paper#;
                  create-alerter A4 %new.reviewer2 %new.paper#;
                  create-alerter A4 %new.reviewer3 %new.paper#",
          creator="Editor"
```

Figure 3(a) Updates of alerter database ADB

```
INSERT (paper#=12,title="Hashing Technique",author="John Doe",
        author-address="12 Main St.,Middletown, IL.",
        submission-date="7/17/1980",paper-status="new") TO PAPERS
```

Figure 3(b) Update of PAPERS relational file

```
To: John Doe
    12 Main St., Middletown, IL.
From: Editor
Subject: Paper submission

This is to acknowledge the receipt of your paper entitled,
"Hashing Technique".
Thank you for your interest in our Journal.
```

Figure 3(c) Form F2

```
To: Editor
From: Editorial Office System
Subject: Reviewer Selection

Please select three reviewers for paper entitled,
"Hashing Technique" by John Doe.

reviewer1:
          -------
reviewer2:
          -------
reviewer3:
          -------
```

Figure 3(d) Form F3

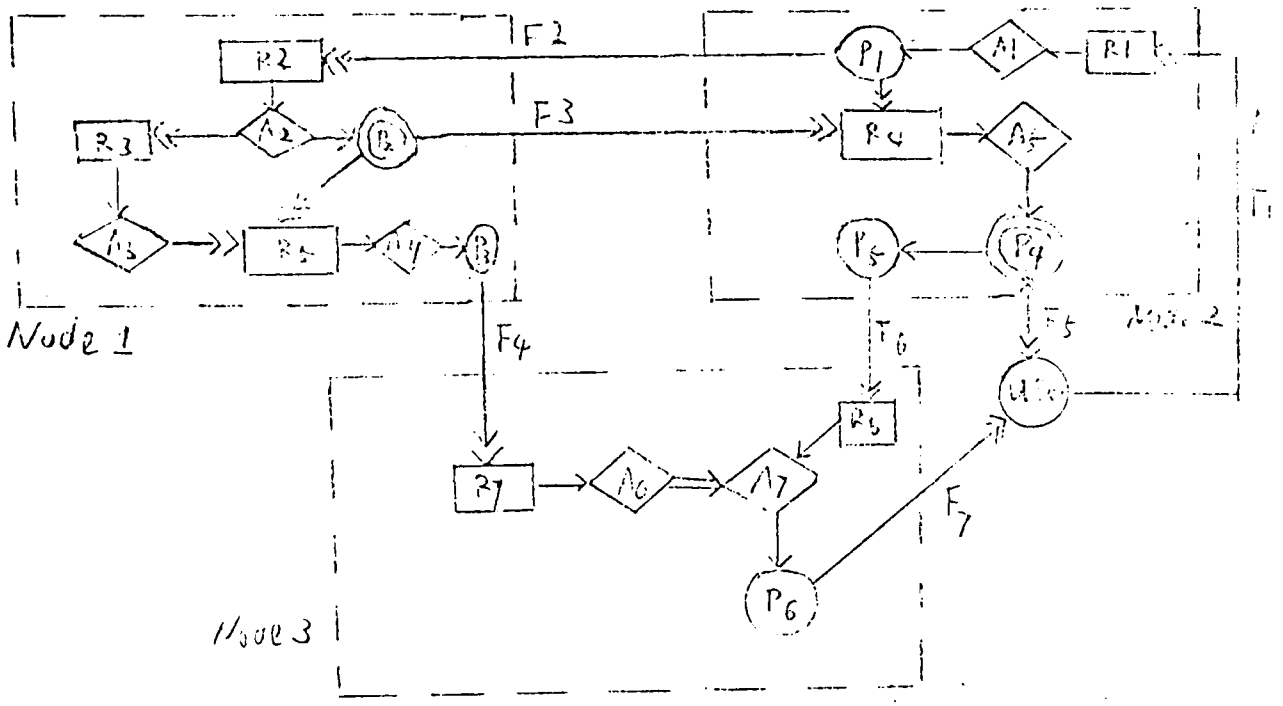


Figure 4: Distributed Office Procedure Model